

1. ATL TRANSFORMATION EXAMPLE: REPLACE INHERITANCE BY ASSOCIATION 1

2. ATL TRANSFORMATION OVERVIEW..... 2

2.1. DESCRIPTION 2

2.2. PURPOSE 2

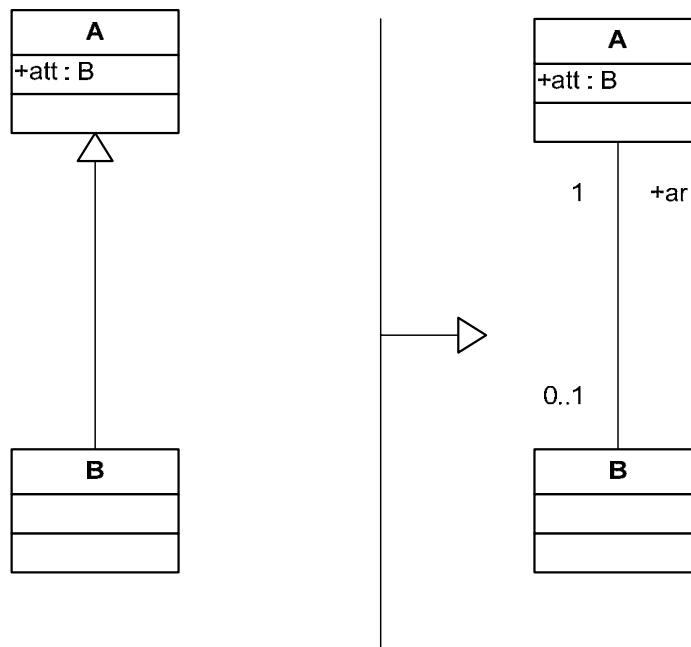
2.3. RULES SPECIFICATION 2

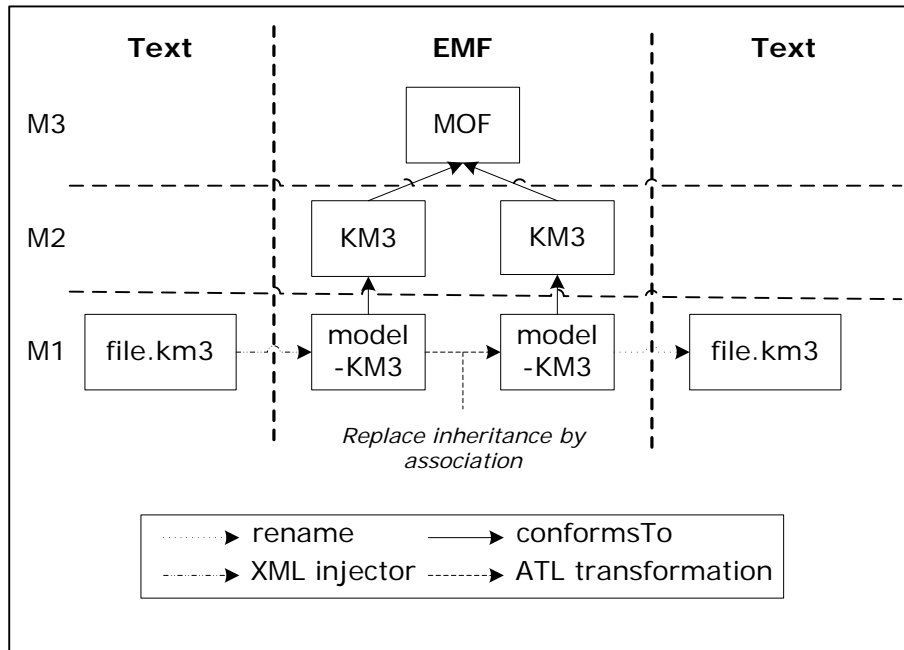
2.4. ATL CODE..... 3

3. REFERENCES 5

1. ATL Transformation Example: replace inheritance by association

This example is extract from [Catalogue of Model Transformations](#) by K. Lano.
Section 1.1: replace inheritance by association, page 2.





2. ATL Transformation overview

2.1. Description

This transformation replaces an inheritance relationship between two classes by an association between the classes.

2.2. Purpose


This transformation is useful when refining a PIM towards a PSM for a platform, which does not support inheritance, such as the relational data model. It can also be used to remove multiple inheritances for refinement to platforms, which does not support multiple inheritances.

Any expression in the original model which has B as contextual classifier, and which uses a feature f inherited from A, must be modified in the new model use ar.f instead.

2.3. Rules specification

Our transformation has the same source and the target metamodel, KM3.

- For a Metamodel element, another Metamodel element is created :
 - with the same name and location,
 - Linked to the same contents.
- For a Package element, another Package element is created :

	ATL Transformation Catalogue of Model Transformations	Author Baudry Julien Jul.baudry <at> gmail.com
	Documentation	Aug 7th 2006

- with the same name,
- Linked to the same contents.
- For a class element, we must distinguish two cases :
 - If the class has no supertypes :
 - We create another class with the same name,
 - Abstract if the source class is abstract.
 - If the class has supertypes :
 - We create another class with the same name,
 - Abstract if the source class is abstract,
 - Without its supertypes, but with a reference to the superclass.

We use a lazy rule to create the association, because an association is composed in two references, and they refer to each other. The superclass has a reference called 'children+ <name of the children>', the other class a reference called 'inherit+ <name of the super class >'. By creating them in the same rule, we can set up the reference opposite correctly.


2.4. ATL Code

```
-- @name      Replacing inheritance by association
-- @version   1.0
-- @domains   Catalogue of Model Transformations
-- @authors   Baudry Julien (jul.baudry<at>gmail.com)
-- @date      2006/08/02
-- @description The purpose of this transformation is to replace inheritance by association
-- @see http://www.dcs.kcl.ac.uk/staff/kcl/tcat.pdf
-- @see section 1.1, page 2
-- @see author of article : K. Lano
```

```
module Replace; -- Module Template
create OUT : KM3target from IN : KM3;
```

```
--@begin rule Metamodel
--entrypoint rule Metamodel() {
-- do {
--   KM3!Metamodel.allInstances().debug('metamodels');
--   KM3!Package.allInstances().debug('packages');
--   KM3!Class.allInstances().debug('classes');
-- }
--}
--@end rule Metamodel

--@begin rule Metamodel
-- For a Metamodel element, another Metamodel element is created :
--   with the same name and location,
--   Linked to the same contents.
rule Metamodel {
  from
    inputMm:KM3!Metamodel
  to
    outputMm:KM3target!Metamodel (
      location <- inputMm.location,
      contents <- inputMm.contents
    )
}
```

	ATL Transformation Catalogue of Model Transformations	Author Baudry Julien Jul.baudry <at> gmail.com
	Documentation	Aug 7th 2006

```

}
--@end rule Metamodel


--@begin rule Package
-- For a Package element, another Package element is created :
--     with the same name,
--     Linked to the same contents.
rule Package {
  from
    inputPkg:KM3!Package
  to
    outputPkg:KM3target!Package (
      name <- inputPkg.name,
      contents <- inputPkg.contents
    )
}
--@end rule Package

--@begin rule Class without supertype
-- This class has no supertypes :
--     We create another class with the same name,
--     Abstract if the source class is abstract.
rule ClassA {
  from
    inputA:KM3!Class (
      inputA.supertypes.isEmpty()
    )
  to
    outputA:KM3target!Class (
      name <- inputA.name.debug('regle 1 '),
      isAbstract <- inputA.isAbstract
    )
}
--@end rule without supertype

--@begin rule Class with supertype
rule ClassB {
  from
    inputB:KM3!Class (
      not(inputB.supertypes.isEmpty())
    )
  to
    outputB:KM3target!Class (
      name <- inputB.name.debug('regle 2 '),
      isAbstract <- inputB.isAbstract,
      structuralFeatures <- inputB.supertypes->iterate(a;acc : Sequence(KM3!StructuralFeature) = Sequence{}|
        acc->including(thisModule.Inherit2Association(a,inputB))
      )
    )
}
--@end rule Class with supertype

lazy rule Inherit2Association {
  from
    supertype:KM3!Class,
    children:KM3!Class
  to
    refChildren : KM3target!Reference (
      name <- 'inherit'+supertype.name,

```

	<p style="text-align: center;">ATL Transformation</p> <p style="text-align: center;">Catalogue of Model Transformations</p>	<p style="text-align: center;">Author</p> <p style="text-align: center;">Baudry Julien Jul.baudry <at> gmail.com</p>
	<p style="text-align: center;">Documentation</p>	<p style="text-align: center;">Aug 7th 2006</p>

```

    opposite <- refSupertype,
    owner <- children
  ),

  refSupertype : KM3target!Reference(
    name <- 'children'+children.name,
    opposite <- refChildren,
    owner <- supertype
  )
}

```

3. References

- [1] Catalogue of Model Transformations
<http://www.dcs.kcl.ac.uk/staff/kcl/tcat.pdf>